# AN ADVANCED SIMULATION FRAMEWORK FOR PARALLEL DISCRETE-EVENT SIMULATION

P Peggy Li, Raymond Yeung,
D'arty Tyrrell, Nadia Adhami,
Tientien Li, t lugh Henry

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, Mail Stop 138-310
Pasadena, California 91109

## ABSTRACT

Discrete-event simulation (DEVS) users have long been faced with a three-way trade-off of balancing execution time, model fidelity, and number of objects simulated. Because of the limits of computer processing power the analyst is often forced to settle for less than desired performance in one or more of these areas, While parallel discrete-event simulation has long been proposed as an approach to overcoming these problems it is seldom used by practicing analysts due to the difficulty of developing parallel simuation code.

This paper describes an environment that supports the simulation analyst in the use of parallel processing technologies in solving application-level simulation problems. This environment, the Advanced Simulation Framework, hides the details of parallel processing from the simulation analyst. This permits the analyst to benefit from the computational power of parallel DEVS (PDEVS) without having to consider the physical reality of a massively parallel platform or network of processors.

## INTRODUCTION

Computer simulation users have always been faced with a requirement to trade-off the scale and scope of studies with their computer's performance and capacity. l-his limitation on performance impacts in several ways. If, as is often the

case, the study uses Monte Carlo or stochastic methods the number of repeated trials must be limited. Additionally, analysts often cannot represent modeled entities at the desired level of detail and/or must restrict the number of entities modeled. Whether limited by insufficient processing power or computer memory, the constraints of computer dependence continue to limit the adequacy of computer-based simulation analysis.

Historically, users have coped by buying larger and more powerful computers or by limiting the scope of their investigations. There is, therefore, a need for an environment for computer simulation that is unconstrained by computer limitations. In such an environment, if an individual computer becomes overloaded the user can merely add another computer or by using more nodes on a massively parallel computer.

The Advanced Simulation Framework addresses this problem through the application of the paradigms o object-oriented technologies to parallel discrete-event simulation (PDEVS). Although parallel processing may seem an obvious solution to inadequate simulation performance, in practice it is surprisingly difficult (Fujimoto 1990). The complexity of programming parallel software discourages many analysts from using this approach. A particular problem is the difficulty of synchronizing the passage of simulated time when multiple computer processes are involved.

To achieve maximum performance in a PDEVS environment, event processing is typically distributed across multiple processors. Events which affect the state of the simulation are communicated with time-star ped messages. Because of the requirement to preserve causality it is essential that messages be processed in nondecreasing time-stamp order. If causality is not preserved a *time accident* occurs. A time accident is an attempt to change the (simulated) past. Most of the difficulty associated with PDEVS comes from the mechanisms used to avoid time accidents.

T hese mechanisms fall into two general categories: *conservative* anti *optimistic.* Conservative PDEVS ensure that time accidents never happen. optimistic PEDS permit time accidents but "clean up" their effects by detection of the accident and *rollback:* a return of the state variables to their pre-accident state. The Advanced Simulation Framework offers both approaches as options to the user.

## FIXED TIME BUCKET ALGORITHM

In this conservative approach, a fixed duration of time, the *time bucket,* is selected by the analyst, The modeling assumption is then that all events occurring within an individual time bucket are independent. In other words, all processing nodes can process all events within the time bucket without fear of time accident. After all processing nodes have completed the time bucket they may exchange messages for newly generated events. If no events are scheduled for a time bucket it is skipped as in a conventional serial DEVS. Note that the time bucket may be any size appropriate to the simulation model; whether a nanosecond or a century, it is merely data.

A drawback of this approach is that processing nodes that finish early (before the slowest node in the configuration) must wait, not doing effective processing. The optimistic *breathing time bucket* algorithm (Steinman 1992) attempts to make use of these "wasted" CPU cycles.

## BREATHING TIME BUCKET ALGORITHM

In the breathing time bucket approach individual processing nodes may process as events as far into the future as desired as long as no event for another nodes is generated. When an external event is generated all nodes must synchronize their simulated time clocks and those nodes that have processed ahead of the time of the new event must rollback to that time. The ASF provides support for incremental saving of simulation object states and rollback to prior states.

The remainder of this paper will describe the implementation of the Advanced Simulation Framework.

## ADVANCED SIMULATION FRAMEWORK OVERVIEW

The Advanced Simulation Framework (ASF) provides distributed simulation services for parallel discrete-event simulation applications running on a network of heterogeneous computers. The ASF is also an object-oriented application that supports object-oriented simulation. It is assumed that the reader is familiar with the fundamental concepts of object-oriented software (Booth 91).

Within this functional context, a simulation application is defined as a set of C++ simulation objects and a scenario. A simulation object consists of attributes that define the state of the object and event methods that define how the object can change states (its own or the states of other objects) .

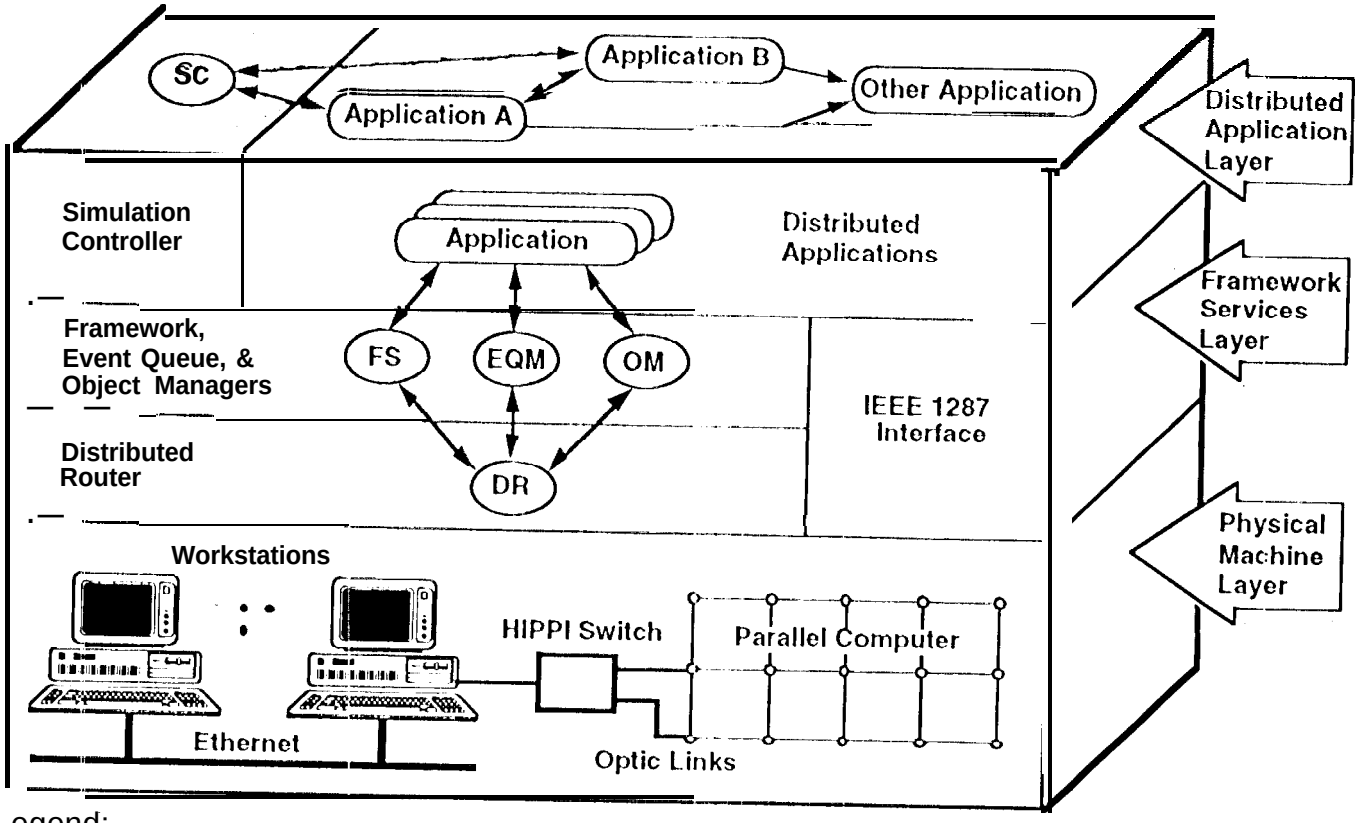
The scenario consists of a configuration, which describes available hardware resources, a description of the initial state of each simulation object, and one or more initial events. When both the set of objects (i. e. the source code) and the scenario are placed within the ASF the simulation applications are executed in a distributed parallel environment.

The system architecture of ASF-based distributed simulations consists of three layers, as shown in Figure 1. At the top of the architecture is the distributed application layer. An ASF-based simulation normally consists of several distributed ASF application modules under the control of the Simulation Controller (SC).

In the middle is the Framework services layer. The Framework provides a library of platform-independent distributed communication, object management, and simulation synchronization and control services. An enhancement to the Framework will also provides standardized interfaces (e.g., Distributed Interactive Simulation (McDonald 93), Aggregate Level Simulation Protocol (MITRE 93)) to allow external applications to be confederated into the simulation.

At the bottom of the architecture is the physical machine layer, which may include Unix-based workstations, such as Sun Sparcstations or Silicon Graphics IRISs connected by a Local or Wide-area Network, and distributed-memory parallel machines, such as the Intel Delta, with high-speed optic channels connected to the rest of the network. The Framework is designed to hide all details of the physical machine and network configuration from the application programmer.

From an application programmer analyst's point of view the ASF consists of a base C++ class C_SIMOBJ, from which simulation objects inherit a variety of methods, and the Simulation Controller. 'l hrough inheritance from the base classes the simulation application gains the services of the Event Queue Manager, the F ramework Server, and the Object Manager. l-he following sections will describe these individual functions of the ASF.




Legend:

SC - Simulation Controller

EQM - Event Queue Manager

DR - Distributed Router

FS - Framework Server

OM - Object Manager

Figure 1. A Layered Software View of an Advanced Simulation Framework-based Simulation

## SIMULATION CONTROLLER AND FRAMEWORK SERVER

The Simulation Controller is the Framework's human user interface to ASF-based simulations. It provides an X-window/Motif-based user interface control panel allowing the user to monitor and control the simulation. From the main control panel the user can perform a variety of functions during each execution or run of the ASF-based simulations including:

- Setting up simulation configuration parameters, initialization values, and various simulation support options.
- Controlling the start and stop of simulation and issue simulation control commands such as check pointing.
- Monitoring application status, error and warning messages, and Framework service utilization.

- Sending commands to simulation objects for "man-in-the-loop" interactivity.

The Simulation Controller performs these functions through interaction with the distributed Framework Servers as described in the next section.

## Framework Server

The simulation controller interacts with individual simulation applications via a network of distributed Framework Servers. The Framework server is a framework system object that resides on a processing node and manages all simulation monitor and control functions on that node. The main functions of the Framework server are:

- Initializing local Framework-based simulation services such as the Object Manager, the Event Queue Manager, and the Distributed Flouter interfaces.
- Maintaining the application's state and status such as initialization parameters, simulation and processing time, and checkpoint and processing status.
- Cooperating with the Simulation Controller to monitor and control the simulation and to maintain a consistent system state.

ASF-based simulation applications do not directly interact with the Framework server. Instead, the Framework provides a set of functions that interface with simulation applications. The following processing steps outline the typical interaction between a simulation application and the Framework server.

Step 1: The simulation application initializes its F ramework support by calling the Framework interface function `simini t`. The Framework Server will initialize the connection to the router, the simulation controller, and all Framework service objects.

Step 2: After the initialization of local Framework support functions, the application can retrieve its node-dependent Framework initialization parameters by calling the Framework interface function `simparam`.

Step 3: After retrieving node-dependent initialization parameters, the application can perform application-specific initialization tasks, such as registering object and event handlers.

Step 4: After completing its application-specific initialization tasks, the application can start its simulation by calling the Framework interface function `sim start` . The F-ramework server will take over the control of simulation from this point on.

Step 5: During the simulation, the Framework server will cooperate with the simulation controller and handle monitor and control requests from the controller

such as *pause, continue, checkpoint,* and *shutdo wn.*

The Framework server is very much like a local agent of the simulation controller. It provides an interface allowing the users to start the simulation, checkpoint/restart the simulation, and to query the state of the simulation via the controller's console.

Event Queue Manager

The Framework provides a distributed-event queue management service via a network of cooperating event queue managers. The Event Queue Manager is a framework system object that resides on a processing node and manages all simulation events on that node. The main functions of the Event Queue Manager are:

- Managing local simulation events activities, such as event creation, processing, and deletion.
- Maintaining an event-driven local simulation time clock and using it to trigger event execution.
- Maintaining processed events and the state change stack to support local rollback.
- Synchronizing and cooperating with other event queue managers to maintain a consistent simulation time space.

ASF-based simulation applications do not directly interact with the Event Queue Manager, Instead, the Framework provides a set of Framework event queue management functions that interface with simulation applications. The following processing steps outline the typical interaction between a simulation application and the Event Queue Manager.

Step 1: A simulation object class is defined, based on the `C_SIMOBJ` class and supplemented with application-specific state variables and methods, including event handling methods, i.e., event handlers.

Step 2: The application registers these event handlers with a Framework Event Queue Manager by calling the Framework interface function `simregisterEventHandler`. After the registration, the Event Queue Manager will add entries to its table to associate the object's type and event type with these event handlers.

Step 3: When the application needs to create a new simulation event, it calls the framework interface function `sim_newEvent` with four event attribute parameters: event time, event type, event recipient, and event content. The Event Queue Manager will add the new event to its queue in the appropriate time order.

Step 4: As determined by the simulation synchronization method used, i.e., fixed time bucket or breathing time bucket, the Event Queue Manager will send external events out and receive its events from other event queue managers at the system synchronization time. The incoming events will be added to its queue in the appropriate time order.

Step 5: When the scheduled execution time of an event is reached, the Event Queue Manager will perform a table lookup for the simulation object and its event handler. The handler will be invoked to process the event with event content passed as an argument.

Step 6: If a processed event needs to be rolled back, as required in the breathing time bucket synchronization scheme, the event manager will pop the state change stack and restore the simulation object to the correct prior state. The processed events and the state change stack will not be released until the next system synchronization time.

Step 7: If the application needs to remove a scheduled event, it can call the Framework interface function `simdeleteEvent` to remove it. l-he Object Manager will look up its queue and remove the event when possible,

The Event Queue Manager is very much like a housekeeper who controls all events on its node and keeps them in the proper order. In addition to providing application interfaces for users to register event handier functions, create new events, and delete existing events, the main tasks of an Event Queue Manager are to maintain the local simulation clock, control the event execution sequence, and perform local roll back if necessary.

## Object Manager

The Framework provides a distributed object management service via a network of cooperating Object Managers. The Object Manager is a framework system object that resides on a processing node and manages all local simulation objects on that node. The main functions of the Object Manager are:

- Managing local simulation objects activities, such as object creation, initialization, and deletion.
- Locating and translating simulation object addresses. Local and remote objects may be referenced by using their name.
- Synchronizing and cooperating with other object managers to maintain a consistent name and address space for simulation objects.

ASF-based simulation applications do not directly interact with the Object Manager. Instead, the Framework

provides a set of functions and a generic simulation object handler class, C_SOHandler, that interface with simulation applications. l-he following processing steps outline the typical interaction between a simulation application and the Object Manager.

Step 1: A simulation object class is defined, based on the c.. SIMOBJ class and augmented with application-specific state variables and methods.

Step 2: A handler for the simulation object class is defined; based on the C_SOHandler class which provides customized access methods for the creation, initialization, and lookup functions.

Step 3: l-he application registers the simulation object handler class with its Object Manager by calling the framework interface function sim_registerObjHandler. After the registration, the Object Manager will add an entry to its table to associate the object's type name with the handler.

Step 4: The application creates a simulation object by calling the framework interface function sim_newObject. The Object Manager will perform a table lookup for the handler and then use the handler's access method to create an object. An entry is also added for the newly created object.

Step 5: If customized initialization is required, the application will call the framework interface function sire. initObject. 1 he Object Manager will perform a table lookup for the object and its handler and then use the handler's access method to initialize the object.

Step 6: Whenever a name is referenced, the Object Manager will perform a table lookup for the object. It may look for the object handler also, if the customized lookup function is required.

Step 7: If the object is no longer needed, the application may call the framework interface function sim_deleteObject to remove it. The Object Manager will perform a table lookup for the object and its handler and then use the handler's access method to delete the object.

The most important function of an Object Manager is to locate simulation objects. By default, the Object Manager uses a table to look up local and remote objects. But, since an application programmer may know how simulation objects are named and distributed, a customized user lookup function for the class of simulation objects can provide a faster and more efficient method for locating objects.

## DISTRIBUTED ROUTER

The framework provides a distributed messaging service via a

network of cooperating Distributed Flouters. T he [distributed Router is a framework system object (not a piece of hardware) that resides on several key processing nodes and manages the routing and delivery of ASF messages. The distributed routers provide a library of communication service functions. These functions allow distributed application modules to do blocking and non-blocking send/receive between dynamically-registered resources.

1'0 use these communication functions, application modules, e.g., framework system objects such as the Object Manager, the Framework Server and the Event Queue Manager, will first register their resource names with their routers. Then communication can be performed by calling functions to send messages to designated resources. The [distributed Router will translate the resource name to addresses and deliver the message to the receiving application or applications. Appropriate message handling functions of the receiving application will be invoked to process the incoming messages.

'1 he communication model used is a unique *dynamic name-based active* communication model. Name-based communication refers to the fact that messages are delivered to a named mail-box, or *resource,* rather than to a physical address. Each Distributed Router maintains a copy of the name-to-address mapping table and uses it to translate resource names to addresses. The name-to-address mapping is a many-to-many function which can map a resource name to zero, one, or more addresses. This concept provides a unified model of commonly used communication methods including: unicast, multicast, and broadcast communication. This concept allows one to separate the logical communication architecture from the physical one, so that simulation applications can be developed based totally on a logical communication architecture.

The name-to-address mapping table can be dynamically modified to reflect changes in the underlying physical communication architecture. Also, the resource name can be dynamically added to or deleted from the mapping table to reflect changes in the logical communication architecture.

The term *active* communication model refers to the fact that after the name-to-address translation, the message is not simply delivered to a holding place at the receiving address, but it is also used to activate a specific handler, or callback method, at that address to process the message automatically.

## SUMMARY

The Advanced Simulation Framework uses the technologies of object-oriented design and programming to permit simulations to be executed in a computationally unconstrained environment. By hiding the technical details of distributed anti parallel simulation the ASF permits the simulation analyst to concentrate his or her energies analytic rather than computer science issues.

## ACKNOWLEDGEMENTS

The authors would like to acknowledge the significant contributions of Dr. Jeffery Steinman, Dr. Yu-Wen Tung and Mr. James Lathrop to the design of the Advanced Simulation Framework. Dr. David Curkendall is the manager of the Advanced Laboratory for Parallel High Performance Applications (ALPI 1A) Project.

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the United States Air Force Electronic Systems Center XR/P "Advanced Programs and Analysis," Hanscom Air Force Base, Massachusetts, through an agreement with the National Aeronautics and Space Administration.

Reference herein to any specific commercial product, process or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government of the Jet Propulsion Laboratory, California Institute of Technology.

## REFERENCES

Booch, G. 1991. *Object-Oriented Design with Applications.* Benjamin Cummings, Redwood City, CA

Fujimoto, R. 1990. "Parallel Discrete Event Simulation. " Communications *of the ACM* 33 no. 10 (Oct.): 30-53.

McDonald, B.; C. Bouwens; and D. Carr, eds. 1992. "Summary Report ,1 he Sixth Workshop on Standards for the Interoperability of Defense Simulations, " Institute for Simulation and T raining University of Central Florida, Orlando, FL. (Mar.)

MITRE Corporation, 1993, "Aggregate Level Simulation Protocol Technical Specification." Mc Lean, VA. (Feb.)

Steinman, J. 1992 "SPEEDES: A Multiple Synchronization Environment for Parallel Discrete-Event Simulation. " *International Journal in Computer Simulation 2, 251-286.*